Deep Learning and Neural Networks: Overview

TRACK MODULE I

Ekaterina Golubeva | ZHAW | Spring 2023

1 Deep Learning and Neural Networks

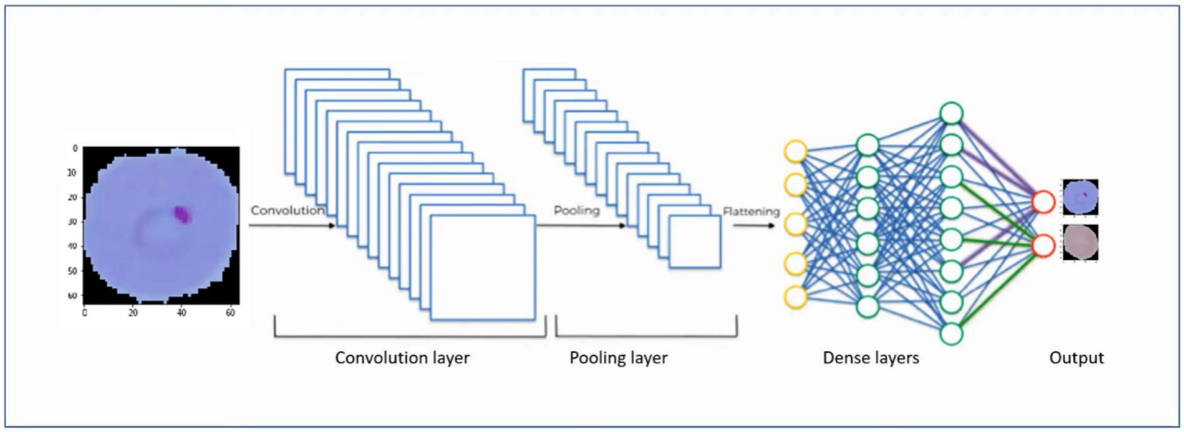
After we've seen traditional ML segmentation (feature generation + classification), let's dive into Deep Learning and Neural Networks. It's not only one type of learning but different depth in terms of learning.

After we provided an input, first level is convolutional : convoluting image with a kernel. Next comes a pooling layer.

Here we have 2 layer, then we have dense layers and gives us an output. It's called deep learning because we can have multiple convolution and pooling layers. This can improve accuracy when we have thousands of training images.

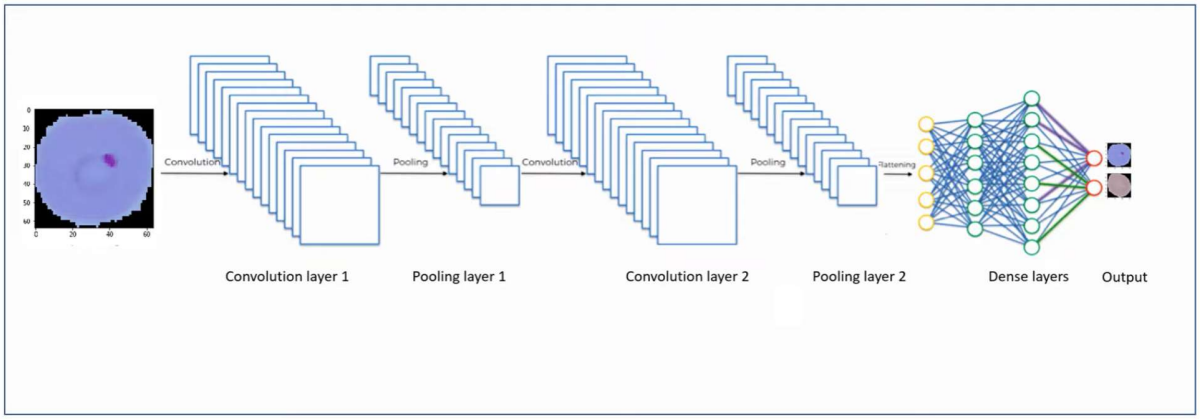
Deep learning (CNN)

Malarial infected vs healthy cell example



Deep learning (CNN)

You can build network with many convolutional and pooling layers



Let’s have a look at some terminology that we’ll need.

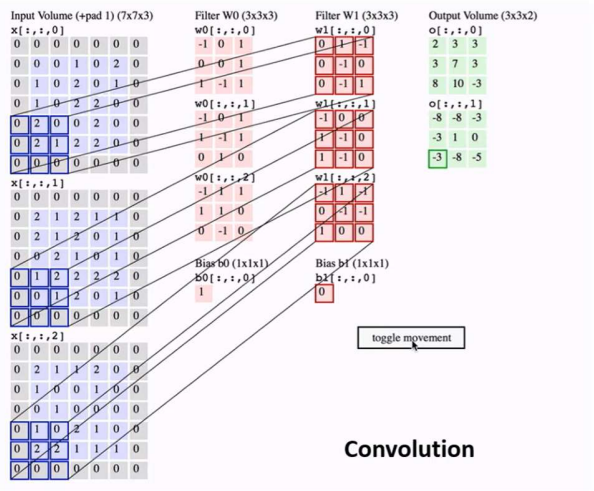
A few keywords to understand:

- Tensorflow
- Keras
- Conv2D
- MaxPooling
- BatchNormalization
- Dropout
- Flatten
- Dense
- Activation = ‘relu’
- Optimizer = ‘adam’
- Loss Function (categorical_crossentropy)

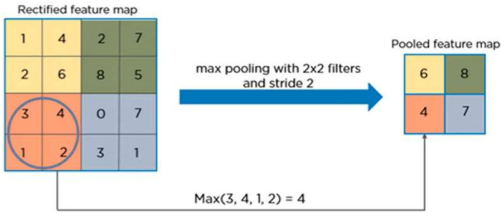
Keras and TensorFlow

TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs (application programming interface) used for easily building and training models, but Keras is more user-friendly because it's built-in Python .

Deep learning (CNN)
Convolution and MaxPooling



Convolution



Max Pooling

Convolution is applying a kernel to an image and we get as an output a matrix of the kernel. We then apply maxpooling which takes the maximum (we can use other sort of poolings e.g average pooling etc).

Deep learning (CNN)

BatchNormalization

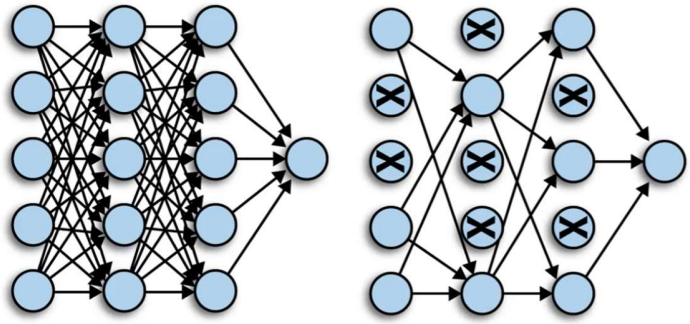
- Normalization is important to keep the values in input and hidden layers within certain range.
- Normalizing usually improves training speed.
- Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.
- We can use higher learning rates because batch normalization makes sure that there's no activation that's gone really high or really low.

We would like to do something between the convolution-pooling steps to enhance the result : we normalize and bring the values to a certain range. It improves the training speed.

Dropout is removing at random non active neurons to prevent overfitting. It's a sort of regularization.

Deep learning (CNN)

Dropout



Standard neural net

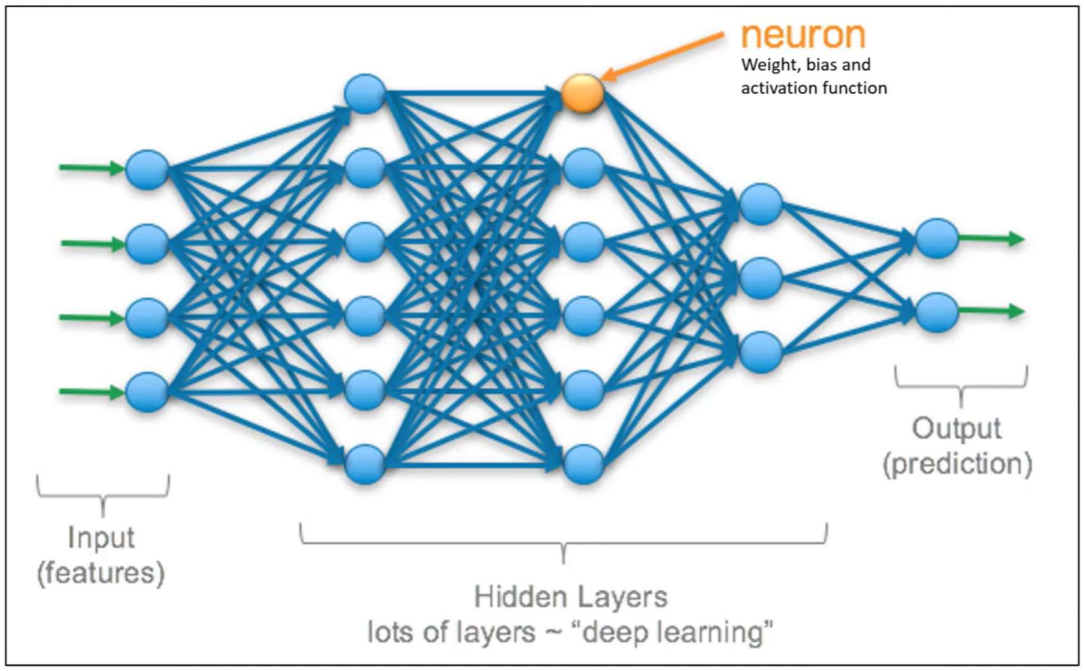
After applying dropout

- The term "dropout" refers to dropping out neurons at random in a neural network.
- Dropout is needed to prevent over-fitting.

Then we flatten the 2D matrices into 1D before going into dense layers. We unwrap the pooled feature map into one array.

In the dense layers, we have neurons. They consist of weights, bias and activation function. Weight is the significance of their contribution to the network, bias is a threshold that determined if the neuron is active or not. The acitivation function decides the threshold and the signal is trasmitted if the theshold is atteigned, otherwise the neuron is not active or dead.

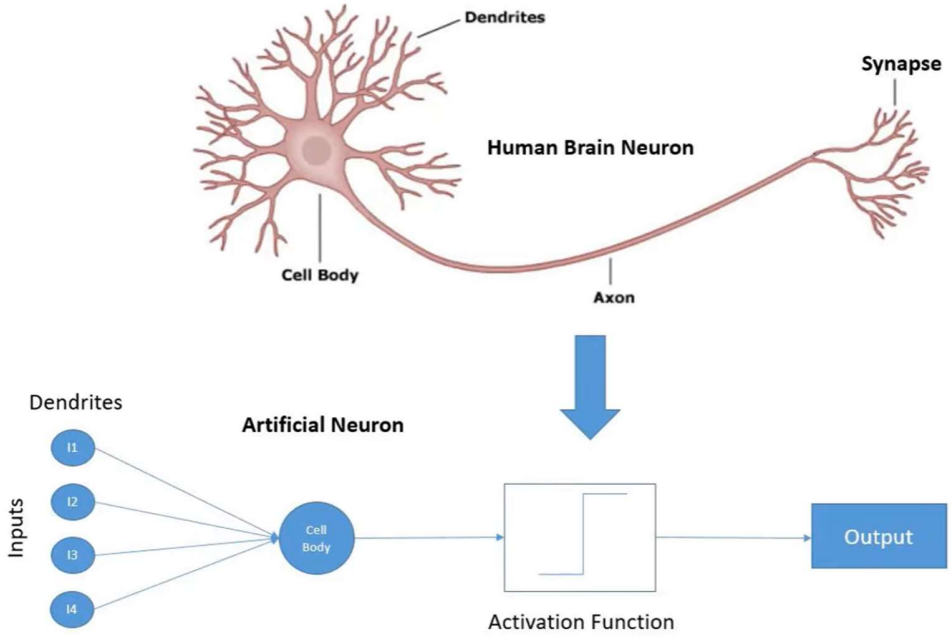
Dense layer



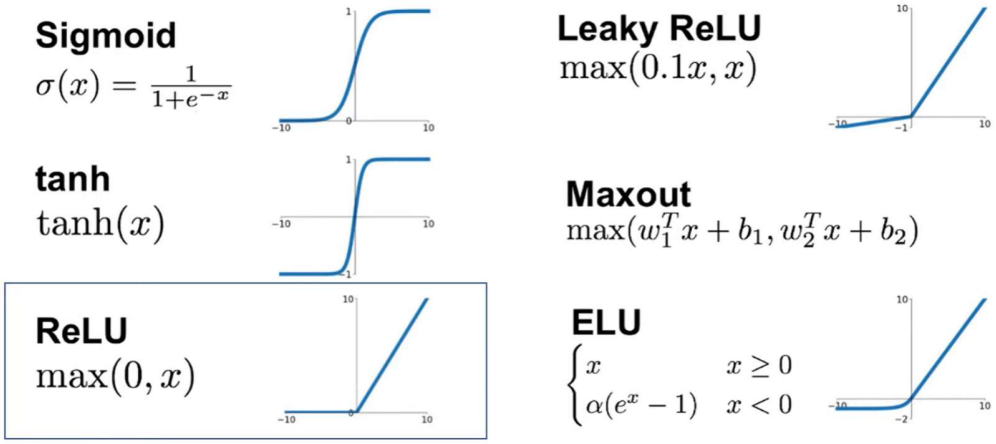
What are neurons ?

An artificial neuron tries to mimic a biological neuron. These are nodes at which decisions are made whether they should transfer the message or not to the further neurons.

Activation: Biological Neuron reference



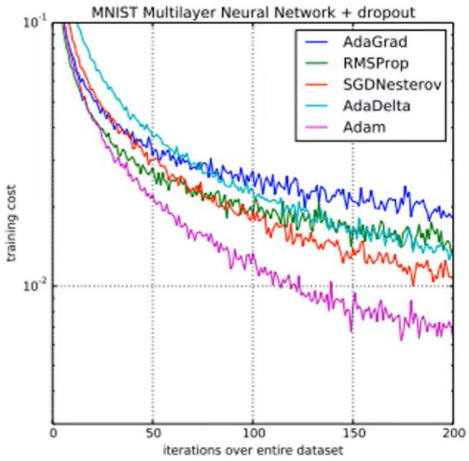
There are different activation funcions :



Another parameter we have to define in our network is the optimizer, it is one of the core parameters of a CNN. We use Adam optimizer which is the extension of the gradient descent that allows us to find the minimum in our loss function. For CNNs we usually will use cross-entropy loss function (research).

Optimizer: Adam

- Optimizer – choice of optimizer algorithm → good results in minutes or hours or days.
- Adam optimization is an extension to stochastic gradient descent.
- Adam is most often used for CNN.



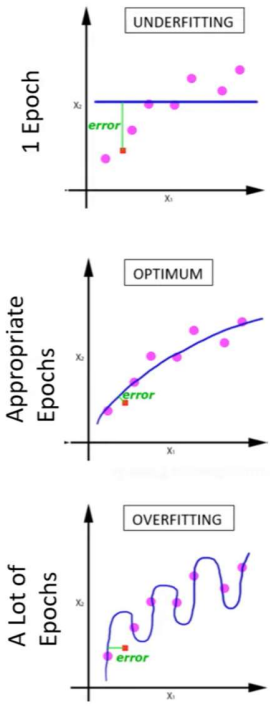
A loss function is a method for evaluating how well our algorithms models the dataset. The samlelr the loss value the better the prediction and vice-versa. Cross-entropy loss function is usually used for CNNs. Mean square error is an example of a basic loss function for linear regression

When the data is too big, we can not download it to the computer but we shouild bring it into smaller batches. Feed forward is the step forward in the dense layer. It's the step at which we get the predicted values and when we can compare them to the actual values. The backward step is when we go back to the neurons and adjust their weights and the activation functions. One forward and one backward for the entire data is called **epoch**.

Deep learning (CNN)

Epoch and other terms

- When data is too big we need to break into smaller batches so the computer can handle.
- One epoch is when an ENTIRE dataset is passed forward and backward through the neural network ONCE.
- Batch Size is the number of training samples in 1 Forward/1 Backward pass.
- Number of iterations = Number of passes
- **Example :** If we have 100 training samples and Batch size is set to 25, it will take 4 iterations to complete 1 Epoch.



If we have similar training examples then we will take usually less epochs than if we had a very diverse set. In general, a greater number of epochs increase the accuracy of the model.

We can now understand the code and implement :

```
INPUT_SHAPE = (64, 64, 3)   #change to (SIZE, SIZE, 3)
inp = keras.layers.Input(shape=INPUT_SHAPE)

conv1 = keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same')(inp)
pool1 = keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)
norm1 = keras.layers.BatchNormalization(axis = -1)(pool1)
drop1 = keras.layers.Dropout(rate=0.2)(norm1)
conv2 = keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same')(drop1)
pool2 = keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
norm2 = keras.layers.BatchNormalization(axis = -1)(pool2)
drop2 = keras.layers.Dropout(rate=0.2)(norm2)
flat = keras.layers.Flatten()(drop2)
hidden1 = keras.layers.Dense(512, activation='relu')(flat)
norm3 = keras.layers.BatchNormalization(axis = -1)(hidden1)
drop3 = keras.layers.Dropout(rate=0.2)(norm3)
hidden2 = keras.layers.Dense(256, activation='relu')(drop3)
norm4 = keras.layers.BatchNormalization(axis = -1)(hidden2)
drop4 = keras.layers.Dropout(rate=0.2)(norm4)

out = keras.layers.Dense(2, activation='sigmoid')(drop4)   #units=1 gives error

model = keras.Model(inputs=inp, outputs=out)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

2 Malarial cell detection using CNNs

We want to classify cells as infected or healthy.

Here's the dataset : https://www.kaggle.com/datasets/iarunava/cell-images-for-detecting-malaria

We'd like to train an algorithm such that, if we give it another dataset, it allows us to predict with a certain confidence if a cell is infected or not.

Workflow :

- Design the network
- Define images (add labels)
- Split dataset into training and testing sets
  (encode labels to_categorical(np.array(label))
- Fit the model
- Plot model accuracy, model.evaluate to monitor CNN performance
- Results
- Testing

The design of our network : convolutional-pooling-normalization-dropout part + dense/hidden layer + flatten = output

Designing the network

From the input layer, we create a convolutional layer, the maxpooling layer(we can create any number of these), normalization and dropouts between the blocks of conv + pooling to minimize overfitting. We then flatten the data and then define a dense layer (we can create any number of these) normalization and dropouts between the dense layers. After that we define the output with activation function = sigmoid. Our loss function is cross-entropy and the optimizer is Adam.

We can also create a fully convolutional network, where there's no dense layer, it's not a neural network.

Model summary :

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0

conv2d (Conv2D)              (None, 64, 64, 32)        896

max_pooling2d (MaxPooling2D  (None, 32, 32, 32)        0
)

batch_normalization (BatchN  (None, 32, 32, 32)        128
ormalization)

dropout (Dropout)            (None, 32, 32, 32)        0

conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248

max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)        0
2D)

batch_normalization_1 (Batc  (None, 16, 16, 32)        128
```

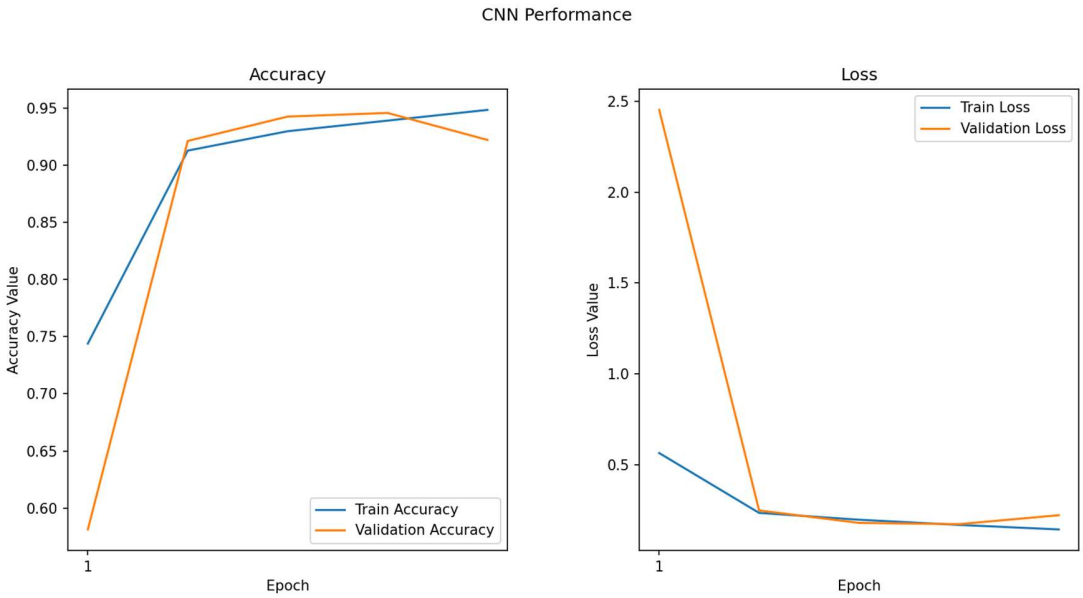
```
 batch_normalization_2 (Batc  (None, 512)               2048
 hNormalization)

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 2)                 514

=================================================================
Total params: 4,340,130
Trainable params: 4,338,466
Non-trainable params: 1,664
```

Fitting the model :

```
Epoch 1/5
311/311 [==============================] - 78s 246ms/step - loss: 0.5643 - accuracy: 0.7439 - val_loss: 2.4528 - val_accuracy: 0.5814
Epoch 2/5
311/311 [==============================] - 76s 243ms/step - loss: 0.2351 - accuracy: 0.9127 - val_loss: 0.2484 - val_accuracy: 0.9211
Epoch 3/5
311/311 [==============================] - 75s 243ms/step - loss: 0.1974 - accuracy: 0.9296 - val_loss: 0.1799 - val_accuracy: 0.9424
Epoch 4/5
311/311 [==============================] - 74s 239ms/step - loss: 0.1690 - accuracy: 0.9389 - val_loss: 0.1731 - val_accuracy: 0.9456
Epoch 5/5
311/311 [==============================] - 77s 248ms/step - loss: 0.1439 - accuracy: 0.9482 - val_loss: 0.2223 - val_accuracy: 0.9220
173/173 [==============================] - 5s 31ms/step - loss: 0.2404 - accuracy: 0.9158
Test_Accuracy: 91.58%
```

Print accuracy test :

With increased number of epochs, one can improve validation accuracy (it should converge).



References :

Code - https://github.com/bnsreenu/python_for_microscopists/blob/master/071-Malaria_cell_CNN_V5.0_for%20video.py

Video on DP and NN - https://youtu.be/LvqzKr-dORQ

Video on Malarial cells detection https://youtu.be/R9PPxpzj5tI

Animation on CNNs - https://cs231n.github.io/convolutional-networks/

Command	Description
inp = keras.layers.Input(shape=INPUT_SHAPE)	Define input
conv=keras.layers.Conv2D(32,kernel_size=(3,3),activation='relu', padding='same')(inp)	Define convolutional layer, activation function, apply to the input layer
pool=keras.layers.MaxPooling2D(pool_size=(2, 2))(conv)	Define pooling layer, apply to the convolutional layer
norm = keras.layers.BatchNormalization(axis=-1)(pool)	Define normalization, apply to previous layer
drop = keras.layers.Dropout(rate=0.2)(norm)	Define dropout, give the percent of data to be dropped, apply to previous layer
flat = keras.layers.Flatten()(drop)	Flatten the matrix to get it ready for dense layer
hidden=keras.layers.Dense(512, activation='relu')(flat)	Defin dense layer, activation function, apply to flattened data
out=keras.layers.Dense(2,activation='sigmoid')(drop4)	Define output from dense layer
model = keras.Model(inputs=inp, outputs=out)	Defin the model using input and output
model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])	Define model parameters such as optimizer, loss function to be minimized at each epoch and evaluation metrics.
print(model.summary()	Model summary gives shape at each layer and number of parameters
history = model.fit(np.array(X_train), y_train, batch_size=64,# 64 images at a tim, verbose=1, epochs=5, # validation_split=0.1, # 10% for validation, shuffle=False)	Fit the model
print("Test_Accuracy:{:.2f}%".format(model.evaluate(np.array(X_test), np.array(y_test))[1] * 100))	Plot accuracy of the model